Nick Wantz

Prof. Shobaki

CSc139-03

3/15/17

a)

1. Array size = 100M, T=2, index for zero =50M+1

|  |  |
| --- | --- |
| Method | Timing (ms) |
| Sequential Search | 43 |
| Parent Waits for All | 44 |
| Parent Busy Wait for All or Quit if 0 | 4 |
| Parent Waits for Semaphore | 2 |

1. Array size = 100M, T=4, index for zero =75M+1

|  |  |
| --- | --- |
| Method | Timing |
| Sequential Search | 63 |
| Parent Waits for All | 45 |
| Parent Busy Wait for All or Quit if 0 | 26 |
| Parent Waits for Semaphore | 16 |

1. Array size = 100M, T=8, index for zero =88M

|  |  |
| --- | --- |
| Method | Timing |
| Sequential Search | 74 |
| Parent Waits for All | 43 |
| Parent Busy Wait for All or Quit if 0 | 24 |
| Parent Waits for Semaphore | 16 |

1. Array size = 100M, T=2, index for zero =-1 (no zero)

|  |  |
| --- | --- |
| Method | Timing |
| Sequential Search | 84 |
| Parent Waits for All | 53 |
| Parent Busy Wait for All or Quit if 0 | 75 |
| Parent Waits for Semaphore | 44 |

1. Array size = 100M, T=4, index for zero =-1 (no zero)

|  |  |
| --- | --- |
| Method | Timing |
| Sequential Search | 84 |
| Parent Waits for All | 50 |
| Parent Busy Wait for All or Quit if 0 | 44 |
| Parent Waits for Semaphore | 37 |

1. Array size = 100M, T=8, index for zero =-1 (no zero)

|  |  |
| --- | --- |
| Method | Timing |
| Sequential Search | 83 |
| Parent Waits for All | 41 |
| Parent Busy Wait for All or Quit if 0 | 57 |
| Parent Waits for Semaphore | 29 |

b)

I wrote and initially tested this program on Titan via ssh. For the timing since I do not have access to the labs, I ran an Ubuntu 14.04 VM on my home PC (I’m sure I could have snuck in but I tested over the weekend). I have an i7 3770 so I figured if I dedicated 4 cores I could get the similar timing results to a true Linux machine. The 3770 specs say that it has 8 possible threads, so I was assuming that it was able to utilize all, but in looking at my results I’m pretty sure that either Windows or VM itself may limit that (not a huge jump in performance from 4 to 8 threads). Even so, based on Amdahl’s law, the performance should have leveled out even with increased access to hardware resources.

The results were as I suspected, from the information learned in class. Sequential search (when there was a zero) increased as the index increased. When there was no zero to find, sequential search timing was essentially the same for every test since it would have had to search all 100M indices.

Parent waiting for all was consistent when there was a zero, since it would have essentially been doing the same task for each test by waiting until all threads searched through their indices. That said there was a moderate performance increase when there was no zero and an increased number of threads. This would point to a direct increase in performance when increasing the number of threads available while utilizing the pthread\_join method.

The busy wait test was very fast in comparison to the previous two when a zero could be found. This is because if a zero was found it closed all threads and returned. What I wasn’t expecting (and I assume can be blamed on my testing environment) was an increase in time as I increased the number of threads. This could be blamed on many things, but I suspect at 4 or 8 threads running through the VM, the first 2 threads got priority and the remaining ones suffered as a result. When there was no zero, as expected it took a lot longer for it to finish compared to both ‘wait for all’ and the semaphore solution. This is due to the fact that there was more work for the parent to do and as a result was competing for resources with the threads.

The semaphore solution was clearly the best of all four. Since the parent process doesn’t compete for CPU resources, and doesn’t have the overhead of pthread\_join where some children may finish before others incurring more parent overhead (thread cleanup). The semaphore solution also seemed to benefit the most from an increase in threads.

This was an interesting experiment to understand the benefits of threading and semaphores, and the clear contrast between various methods of process synchronization.